Inner Product Masking Revisited

Josep Balasch[1(✉)], Sebastian Faust[2,3], and Benedikt Gierlichs[1]

[1] KU Leuven Department Electrical Engineering-ESAT/COSIC and iMinds, Kasteelpark Arenberg 10, B-3001 Leuven-Heverlee, Belgium
{Josep.Balasch,Benedikt.Gierlichs}@esat.kuleuven.be
[2] EPFL Lausanne, Lausanne, Switzerland
[3] Ruhr-University Bochum, Bochum, Germany
sebastian.faust@gmail.com

Abstract. Masking is a popular countermeasure against side channel attacks. Many practical works use Boolean masking because of its simplicity, ease of implementation and comparably low performance overhead. Some recent works have explored masking schemes with higher algebraic complexity and have shown that they provide more security than Boolean masking at the cost of higher overheads. In particular, masking based on the inner product was shown to be practical, albeit not efficient, for a small security parameter, and at the same time provable secure in the domain of leakage resilient cryptography for a large security parameter. In this work we explore a security versus efficiency tradeoff and provide an improved and tweaked inner product masking. Our practical security evaluation shows that it is less secure than the original inner product masking but more secure than Boolean masking. Our performance evaluation shows that our scheme is only four times slower than Boolean masking and more than two times faster than the original inner product masking. Besides the practical security analysis we prove the security of our scheme and its masked operations in the threshold probing model.

1 Introduction

Side-channel attacks (SCA) are a well-known threat to embedded security. They allow to perform key recovery attacks on cryptographic implementations by analyzing physical properties present in embedded devices. Examples are execution time [22], power consumption [23] or electromagnetic emanations [16,30]. SCA exploit the fact that these measurable quantities are statistically dependent on the intermediate variables being processed in the implementation. One of the most popular and well-studied countermeasures for block ciphers are data randomization techniques, commonly known as masking [5,19]. These aim to conceal all intermediate variables of a cryptographic computation with random data.

The core principle of higher-order masking is to split any sensitive variable S into n random and secret shares. The way in which such a splitting is made determines the *masking type* or the *masking function*. Typical examples are Boolean masking ($S = S_1 + \ldots + S_n$) or multiplicative masking ($S = S_1 \times \ldots \times S_n$). A

E. Oswald and M. Fischlin (Eds.): EUROCRYPT 2015, Part I, LNCS 9056, pp. 486–510, 2015.
DOI: 10.1007/978-3-662-46800-5_19

masking scheme additionally defines a set of operations to process the n shares while preserving the correctness of computations and ensuring that intermediate values remain independent of the sensitive variables. The *security order* d of a masking scheme is defined as the smallest number of $d+1$ intermediate values that, considered jointly, are *not* independent of a sensitive variable S. While a d-th order masking scheme can always be broken by a $d+1$-th order SCA that exploits the leakage of $d+1$ intermediate values in the protected implementation jointly, the design of higher-order masking countermeasures is of practical interest due to two main reasons. First, the data complexity of applying a $d+1$-th order SCA grows exponentially on d given a sufficient amount of noise [5,11,27]. And second, the computational complexity of searching for the $d+1$ leakage points grows combinatorially in the attack order [32].

1.1 Related Work

Ishai, Sahai and Wagner [21] were first to introduce a higher-order Boolean masking scheme tailored to hardware contexts by showing how to protect a circuit over $\mathbb{F}_2$ composed of NOT and AND gates. They also proved the security of their construction against an adversary capable of probing d wires of the protected circuit. The framework introduced in [21] (commonly known as the ISW probing model) provides a sound basis to determine the security of higher-order masking schemes, as security against d probes directly implies d-th order SCA resistance [7]. Ishai *et al.* proved their construction secure in the probing model for $n \geq 2d+1$.

Rivain and Prouff [33] extended the ideas of [21] to any finite field. They devised a series of provable secure operations in the masked domain and applied them to efficiently secure AES implementations in software contexts. To protect the most complex part of AES, namely the nonlinear part of the S-box, they employed a power function devised to minimize the number of costly multiplications. Although the original claim in [33] was that $n = d+1$ shares could provide d-th order provable security, Coron *et al.* [9] have shown that in fact $n \geq 2d+1$ shares are necessary. Despite this, the Boolean scheme due to Rivain and Prouff remains one of the most efficient generic higher-order constructions in the literature.

Further work has focused on improving the performance of higher-order Boolean masking for the most challenging part of cipher implementations, namely nonlinear transformations. Genelle *et al.* [17] proposed a method to securely switch between Boolean and multiplicative masking at any order. The technique is particularly suitable to protect implementations of the AES, as it enables to switch the mask type before and after the S-box power function. Carlet *et al.* [4] built on the ideas of [33] to secure any look-up table using Lagrange interpolation. Coron [7] introduced a method to mask look-up tables at any order.

Generic higher-order countermeasures using other types of masking have also been investigated. Because of their higher algebraic complexity, observation of the shares results in significantly less information leakage than for the Boolean type given the same security order d and low levels of noise. Along these lines,

von Willich [37] proposed *affine* masking in which variables are encoded as $S' = \text{mask}_1 \times S + \text{mask}_2$. Fumaroli *et al.* [15] analyzed this type of masking and showed how it can be used to secure AES software implementations, in which the S-box is protected by means of table re-computation. The authors achieve performance results that are comparable to those of Boolean masking, but the affine masking construction has not been generalized to higher security orders $d > 1$.

Another proposal for a different type of masking is *polynomial* masking. Independently introduced by Prouff and Roche [29] as well as by Goubin and Martinelli [18], it employs Shamir's secret-sharing [34] and secure multi-party computation techniques [2]. In particular, a sensitive variable S is associated to a polynomial of degree d of the form $P_S(X) = S + \sum_{i=1}^{d} a_i \times X^i$ where the a_i are random secret coefficients. An encoding of a variable S is performed by selecting n distinct nonzero elements α_i and evaluating $S_i = P_S(\alpha_i)$ for $i = 1, \ldots, n$. The variable S is then represented in the masked domain as the combination of n pairs (α_i, S_i), of which only the S_i are secret. The variable S can be reconstructed as $S = \sum_{i=1}^{n} S_i \times \beta_i$, where the coefficients β_i are computed from the public α_i.

Although based on the same masking type, the schemes due to Prouff and Roche [29] and by Goubin and Martinelli [18] have notable differences. In particular, the construction of [29] is specifically designed to prove d-th order security in the presence of *glitches* [24], while the construction of [18] is designed to achieve "classical" d-th order SCA resistance. Both schemes use an algorithm to compute the product of two masked variables that is based on the secure multi-party computation scheme due to Ben-Or *et al.* [2], and that requires $n \geq 2d+1$. The authors of [18] additionally propose a more efficient algorithm that reduces the number of required shares to $n \geq d + 1$. Despite this improvement, the complexity of both multiplication algorithms is $\mathcal{O}(n^3)$ in the number of shares, as opposed to e.g. Boolean masking which achieves $\mathcal{O}(n^2)$. A recent work by Coron *et al.* [8] has improved this complexity to $\tilde{\mathcal{O}}(n^2)$ by using DFT for fast polynomial evaluation.

A different approach is followed by Balasch *et al.* [1]. They introduce *IP* masking based on the inner product construction of Dziembowski and Faust [12]. A masked variable is represented by $2n$ shares in the form of two random vectors $(\boldsymbol{L},\boldsymbol{R})$ of n elements each such that S equals the inner product of $\boldsymbol{L}$ and $\boldsymbol{R}$. The authors propose an efficient multiplication algorithm with complexity $\mathcal{O}(n^2)$, thus similar to Boolean constructions, while achieving significantly less information leakage than other types of masking at the same security order d for low noise levels. Despite these advantages, the addition and refreshing algorithms are more complex (larger constant terms) than their counterparts in the Boolean and polynomial masking schemes.

Overall, the applicability of higher-order masking techniques other than Boolean is still an open question. Coron *et al.* [8] have identified a first-order flaw in the faster multiplication routine of the polynomial masking construction of Goubin and Martinelli [18]. Similarly, a first-order leakage in the refresh and addition operations of IP Masking [1] has been pointed out by Prouff *et al.* [28]. And the polynomial masking scheme by Prouff and Roche [29] has been shown

to be rather demanding when implemented in hardware [25], mostly due to its security guarantees even in the presence of glitches. Eventually, and despite their potential, higher-order countermeasures based on masking constructions other than Boolean do not appear to be ready for practical applications.

1.2 Our Contributions

In this work, we develop several improvements to the original IP masking scheme proposed in [1].

New IP Masking Scheme. Our first contribution is to introduce a few tweaks in the definition of the masking function that result in significant performance improvements. Similar to the original IP masking, a masked variable is represented by $2n$ shares in the form of two vectors ($\boldsymbol{L}$,$\boldsymbol{R}$). Our first change is to let $\boldsymbol{L}$ be a public value, allowing us to reduce the number of secret shares from $2n$ to n. As a result we fix $\boldsymbol{L}$ to a constant value in such a way that all variables involved in computations are masked under the same $\boldsymbol{L}$, but different $\boldsymbol{R}$. Additionally, we require that the first public element of $\boldsymbol{L}$ is $L_1 = 1$. The combination of these changes results in great efficiency improvements for all operations in the masked domain. In particular, the complexity of the addition and refreshing algorithms becomes comparable to those of Boolean and polynomial masking schemes. An important side benefit of our tweaks is that the first-order leakage identified in [28] on the refresh and addition algorithms no longer applies.

Practical Security Analysis. Our second contribution is to evaluate the impact of our tweaks and compare the security of the new masking type with other higher-order masking functions. We use the mutual information between the secret variable and the leakage of all shares of its masked representation as figure of merit. Our evaluation shows that our new masking function leaks more than IP masking, which is expected because $\boldsymbol{L}$ is now public, but it leaks roughly one order of magnitude less than Boolean masking with the same security order d. It also leaks similar to polynomial masking with the same security order d.

Security in the Threshold-Probing Model. As a third contribution we prove the security of our improved scheme in the probing model introduced by Ishai, Sahai and Wagner [21]. Our security analysis shows that our construction is secure against d probes, when $n \geq 2d + 1$. We emphasize that this is the same security threshold that can be achieved by most other higher-order masking schemes [7, 21,33]. Notice that only for the multiplication operation we require that $n \geq 2d + 1$. For all other operations including the masking function, it is sufficient to set $n > d$.

Efficient Implementation. Finally, our fourth contribution is to determine the performance of our masking scheme in securing a block cipher implementation. Similar to [1], we opt to protect a software implementation of AES-128 on an

embedded 8-bit controller and we compare our results to other d-th order masking schemes. The results show that our improved construction allows to halve the execution time with respect to the original IP masking scheme, and to reduce the gap with Boolean masking from approximately a factor 10 to a factor 4.

2 Notation

In the following we denote by $\mathcal{K}$ a field of characteristic 2 and we represent field elements with upper-case letters. For instance, $S \in \mathbb{F}_{2^8}$ denotes an element in the AES field $GF(2^8)$. Let $\boldsymbol{X}, \boldsymbol{Y}$ represent two vectors over $\mathcal{K}^n$. Field elements in vectors are addressed with subindex i, e.g. X_i and Y_i, respectively. The standard inner product function over $\mathcal{K}$ is denoted as $\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \sum_i X_i \times Y_i$.

The matrix $\hat{\boldsymbol{A}} = \boldsymbol{X} \times \boldsymbol{Y}$ over $\mathcal{K}^{n \times n}$ is defined as the tensor product of two vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$. Field elements in a matrix are addressed as $A_{i,j}$, where i and j represent row and column position, respectively. If $\hat{\boldsymbol{A}}, \hat{\boldsymbol{B}}$ are matrices over $\mathcal{K}^{n \times n}$, then we denote by $\langle \hat{\boldsymbol{A}}, \hat{\boldsymbol{B}} \rangle$ the inner product of matrices when we view them as vectors of n^2 field elements, i.e. $\sum_i \sum_j A_{i,j} \times B_{i,j}$.

For an integer n we denote by $[n]$ the set $\{1, \ldots, n\}$.

3 Our Construction

Our scheme improves on the IP masking by Balasch *et al.* [1] based on the inner product construction of Dziembowski and Faust [12]. A variable $S \in \mathcal{K}$ in IP masking is encoded by using two vectors $(\boldsymbol{L}, \boldsymbol{R})$ of n elements with $\boldsymbol{L} \leftarrow \mathcal{K} \backslash \{0\}^n$ and $\boldsymbol{R} \leftarrow \mathcal{K}^n$. Note that the elements of the vector $\boldsymbol{L}$ are by definition different than zero.

Our new masking function has three major differences with respect to [1]. First, the vector $\boldsymbol{L}$ is computed once and kept as a constant parameter. This implies that all masked variables employed in our scheme share a unique vector $\boldsymbol{L}$. Second, we let the vector $\boldsymbol{L}$ be a public (rather than secret) parameter. In other words, we assume the elements L_i can be known to the adversary. And third, we constrain the selection of the first element of $\boldsymbol{L}$ such that $L_1 = 1$. The number of shares that are kept secret in our masking function is therefore determined by the security parameter n, which corresponds to the number of elements in $\boldsymbol{R}$. We show in the remainder of this section that all these choices allow to significantly reduce the complexity of operations in the masked domain.

The procedure $\texttt{IPSetup}_n$ depicted in Algorithm 1 details the initialization steps of our masking construction. Given n and a field description $\mathcal{K}$, the algorithm returns a public vector $\boldsymbol{L}$ and a public matrix $\hat{\boldsymbol{L}}$. The latter corresponds to the tensor product $\boldsymbol{L} \times \boldsymbol{L}$, and its pre-computation allows us to speed-up multiplications in the masked domain. One can imagine $\texttt{IPSetup}_n$ is executed before system roll-out during device personalization, e.g. in parallel with key generation. The sub-routine `randNonZero()` returns a nonzero element in the field $\mathcal{K}$. More precisely, it samples uniformly at random from $\mathcal{K} \backslash \{0\}$.

Algorithm 1. Setup the masking scheme: $(\boldsymbol{L}, \hat{\boldsymbol{L}}) \leftarrow \texttt{IPSetup}_n(\mathcal{K})$

Input: field description $\mathcal{K}$
Output: random vector $\boldsymbol{L}$ and tensor product $\hat{\boldsymbol{L}} = \boldsymbol{L} \times \boldsymbol{L}$

1: $L_1 = 1$;
2: **for** $i = 2$ to n **do**
3: $\quad L_i \leftarrow \texttt{randNonZero}(\mathcal{K})$;
4: **end for**
5: **for** $i = 1$ to n **do**
6: $\quad$ **for** $j = 1$ to n **do**
7: $\quad\quad L_{i,j} = L_i \times L_j$;
8: $\quad$ **end for**
9: **end for**

Algorithm 2 depicts the steps to convert a variable $S \in \mathcal{K}$ into the masked domain. This routine, denoted by $\texttt{IPMask}_{\boldsymbol{L}}$, is parametrized by a vector $\boldsymbol{L}$ resulting from executing $\texttt{IPSetup}_n$. The sub-routine $\texttt{rand}()$ returns a randomly selected element in the field $\mathcal{K}$. This function is called $n-1$ times in order to set the values of $R_2 \ldots R_n$. The value R_1 is then computed in order to obtain a valid masking of S under the inner product construction.

Algorithm 2. Masking a variable: $\boldsymbol{R} \leftarrow \texttt{IPMask}_{\boldsymbol{L}}(S)$

Input: variable $S \in \mathcal{K}$
Output: vector $\boldsymbol{R}$ such that $S = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$

1: **for** $i = 2$ to n **do**
2: $\quad R_i \leftarrow \texttt{rand}(\mathcal{K})$;
3: **end for**
4: $R_1 = S + \sum_{i=2}^{n} L_i \times R_i$

3.1 Operations in the Masked Domain

We propose three main algorithms of our masking construction: $\texttt{IPRefresh}_{\boldsymbol{L}}$, $\texttt{IPAdd}_{\boldsymbol{L}}$ and $\texttt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}$. For an implementation of the AES (as detailed in Sect. 6) it is advantageous to further have a dedicated squaring routine in the masked domain. For this reason we also propose an additional $\texttt{IPSquare}_{\boldsymbol{L}}$ algorithm.

The routine $\texttt{IPRefresh}_{\boldsymbol{L}}$ is depicted in Algorithm 3. It consists of two steps. First, the computation of a vector $\boldsymbol{A}$ orthogonal to $\boldsymbol{L}$. And second, the addition of $\boldsymbol{A}$ to $\boldsymbol{R}$ to obtain a fresh vector $\boldsymbol{R'}$. The correctness of the algorithm is easy to prove:

$\langle \boldsymbol{L}, \boldsymbol{R'} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} + \boldsymbol{A} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} \rangle + \langle \boldsymbol{L}, \boldsymbol{A} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} \rangle + 0 = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$.

The routine $\texttt{IPAdd}_{\boldsymbol{L}}$ is illustrated in Algorithm 4. Because all variables in our construction are masked with a common vector $\boldsymbol{L}$, the output vector $\boldsymbol{T}$ can be simply obtained by adding the input vectors $\boldsymbol{R}$ and $\boldsymbol{Q}$.

Algorithm 3. Refresh vector: $\boldsymbol{R}' \leftarrow \texttt{IPRefresh}_{\boldsymbol{L}}(\boldsymbol{R})$

Input: vector $\boldsymbol{R}$
Output: vector $\boldsymbol{R}'$ such that $\langle \boldsymbol{L}, \boldsymbol{R} \rangle = \langle \boldsymbol{L}, \boldsymbol{R}' \rangle$
1: $\boldsymbol{A} \in_R \mathcal{K}^n$ s.t. $\langle \boldsymbol{A}, \boldsymbol{L} \rangle = 0$
2: $\boldsymbol{R}' = \boldsymbol{R} + \boldsymbol{A}$

Algorithm 4. Add masked values: $\boldsymbol{T} \leftarrow \texttt{IPAdd}_{\boldsymbol{L}}(\boldsymbol{R}, \boldsymbol{Q})$

Input: vectors $\boldsymbol{R}$ and $\boldsymbol{Q}$
Output: vector $\boldsymbol{T}$ such that $\langle \boldsymbol{L}, \boldsymbol{T} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} \rangle + \langle \boldsymbol{L}, \boldsymbol{Q} \rangle$
1: $\boldsymbol{T} = \boldsymbol{R} + \boldsymbol{Q}$

Note that the $\texttt{IPAdd}_{\boldsymbol{L}}$ algorithm is similar to that of Boolean masking constructions, yet our type of masking has higher algebraic complexity. This improvement is a direct consequence of letting $\boldsymbol{L}$ be a public and constant parameter.

The multiplication routine $\texttt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}$ depicted in Algorithm 5 is the most involved operation. Our starting point is the masked multiplication of [1], albeit with some efficiency improvements. First, since both input operands are masked under the same vector $\boldsymbol{L}$, the computation of the matrix $\hat{\boldsymbol{L}}$ is not dependent on the input operands. Consequently, we can save n^2 field multiplications by pre-computing this matrix during $\texttt{IPSetup}_n$. And second, because the first component of $\boldsymbol{L}$ is set to $L_1 = 1$, a constant b can be added to a masking $(\boldsymbol{L}, \boldsymbol{R})$ by simply computing the new masking as $(\boldsymbol{L}, (R_1 + b, R_2, \ldots, R_n))$.

Algorithm 5. Multiply masked values: $\boldsymbol{T} \leftarrow \texttt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}(\boldsymbol{R}, \boldsymbol{Q})$

Input: vectors $\boldsymbol{R}$ and $\boldsymbol{Q}$
Output: vector $\boldsymbol{T}$ such that $\langle \boldsymbol{L}, \boldsymbol{T} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} \rangle \times \langle \boldsymbol{L}, \boldsymbol{Q} \rangle$
1: $\hat{\boldsymbol{A}} \in_R \mathcal{K}^{n \times n}$ s.t. $\langle \hat{\boldsymbol{L}}, \hat{\boldsymbol{A}} \rangle = 0$
2: $\hat{\boldsymbol{R}} = \boldsymbol{R} \times \boldsymbol{Q}$
3: $\hat{\boldsymbol{B}} = \hat{\boldsymbol{R}} \oplus \hat{\boldsymbol{A}}$
4: $b = \sum_{i=2}^{n} \sum_{j=1}^{n} L_{i,j} \times B_{i,j}$
5: $\boldsymbol{T} = (B_{1,1} + b, B_{1,2}, \ldots, B_{1,n})$

In order to keep the d-th order security for $n = 2d + 1$ throughout the whole execution of $\texttt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}$, it is important that operations in lines `1` and `4` are computed in a certain way as depicted in Table 1. In particular, the intermediate values Δ_j are calculated by aggregating the intermediate products of elements in matrices $\hat{\boldsymbol{A}}$ and $\hat{\boldsymbol{L}}$ in a column-wise fashion. In contrast, the values β_i are computed by processing the elements of the matrices $\hat{\boldsymbol{L}}$ and $\hat{\boldsymbol{B}}$ row by row. This important difference in the way we compute the sums is crucial for the security proof and, in fact, crucial for the actual security of our scheme.

For illustration purposes, consider a setting where the operations in line 1 and line 4 of $\texttt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}$ are computed row-wise. Let us also assume that all elements in $\boldsymbol{L}$ are $L_i = 1$. Under these circumstances, the following attack would apply:

Table 1. Detailed description of operations in $\texttt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}$

LINE 1: $\hat{\boldsymbol{A}} \in_R \mathcal{K}^{n\times n}$ s.t. $\langle \hat{\boldsymbol{L}}, \hat{\boldsymbol{A}} \rangle = 0$	LINE 4: $b = \sum_{i=2}^{n} \sum_{j=1}^{n} L_{i,j} \times B_{i,j}$
// Random sampling	*// Row-wise processing*
for $(i,j) \neq (n,n)$ **do**	$\beta_1 = 0$
$A_{i,j} \leftarrow \texttt{rand}(\mathcal{K})$	**for** $i = 2$ to n **do**
end	$\beta_i = \beta_{i-1} + \sum_{j=1}^{n} L_{i,j} \times B_{i,j}$
// Column-wise processing	**end**
$\Delta_0 = 0$	$b = \beta_n$
for $j = 1$ to $n-1$ **do**	
$\Delta_j = \Delta_{j-1} + \sum_{i=1}^{n} A_{i,j} \times L_{i,j}$	
end	
$A_{n,n} = (\Delta_{n-1} + \sum_{i=1}^{n-1} A_{i,n} \times L_{i,n}) \times L_{n,n}^{-1}$	

1. The adversary learns the value $\Delta_2 = \sum_{j=1}^{n} A_{2,j}$
2. The adversary learns $\beta_2 = \sum_{j=1}^{n} A_{2,j} + Q_2 \times R_j = \Delta_2 + Q_2 \langle \boldsymbol{L}, \boldsymbol{R} \rangle$
3. The adversary learns Q_2

Assuming that $Q_2 \neq 0$ the above attack indeed recovers completely the secret value $\langle \boldsymbol{L}, \boldsymbol{R} \rangle$. Notice that the attack even applies when $L_i \neq 1$ but in this case the bias in the leaky distribution decreases with the number of shares. We prevent this attack by computing the intermediate values Δ_j as a sum of elements in a column, and β_i as a sum of elements in a row. This approach effectively results in a "mixing" of the random shares and enables a security proof.

For an implementation of the AES it is beneficial to have a particularly efficient implementation of the squaring algorithm. The $\texttt{IPSquare}_{\boldsymbol{L}}$ routine is illustrated in Algorithm 6.

Algorithm 6. Square masked variable: $\boldsymbol{T} \leftarrow \texttt{IPSquare}_{\boldsymbol{L}}(\boldsymbol{R})$

Input: vector $\boldsymbol{R}$
Output: vector $\boldsymbol{R}'$ such that $\langle \boldsymbol{L}, \boldsymbol{T} \rangle = \langle \boldsymbol{L}, \boldsymbol{R} \rangle \times \langle \boldsymbol{L}, \boldsymbol{R} \rangle$

```
for i = 1 to n do
    T_i ← (R_i)^2 × L_i;
end for
```

4 Practical Evaluation

In this section we evaluate the information leakage of our improved IP masking scheme and compare it to that of other masking schemes that can be implemented at any order, e.g. Boolean masking, polynomial masking and the original IP masking. We follow the common approach to focus the analysis on the type of masking, i.e. to analyze the leakage of the shares of one masked variable. In practice, the leakage of the operations in the masked domain depends a lot on

their (secure) implementation and on the target platform, for instance a table lookup in software versus combinational logic in hardware. We abstract from such practical issues to be able to provide a fair and meaningful comparison.

4.1 Attack Order

We begin the evaluation by deriving the minimum order for an attack against our type of masking. We say that a masking function is d-th order SCA secure, if every tuple of d or less shares is independent of the masked variable.

It is easy to see that the masking function of our improved scheme is d-th order SCA secure for $n = d + 1$. If we choose all elements in $\boldsymbol{L}$ equal to 1 the argument is exactly the same as for Boolean masking. It uses the same number of uniformly distributed secret shares. If we choose any L_i greater than 1 we just need to observe that the field multiplication $L_i \times R_i$ does not introduce biases, i.e. for uniformly distributed R_i the output is uniformly distributed. Recall that all elements in $\boldsymbol{L}$ are nonzero by definition.

4.2 Information Leakage

It remains to explore the security versus efficiency tradeoff of our improved scheme. It is known that a more complex algebraic relation between the shares and the masked variable provides less information leakage. We hence expect our type of masking to leak less information than the Boolean type whenever at least one L_i is unequal to 1 (since the field multiplication $L_i \times R_i$ adds diffusion) but more than the original IP masking since $\boldsymbol{L}$ is public.

Similar to our type of masking, in polynomial masking half of the shares are distinct nonzero public constants and the other half are random and secret masks. For our evaluation of information leakage we refer only to the n secret shares. For example, polynomial masking of security order d uses $n = d + 1$ random and secret shares, and can theoretically be broken by a $d + 1$-th order SCA. Due to the similar representations of variables in the masked domain, we expect our masking function and the polynomial type to provide comparable information leakage.

We compare our type of masking with Boolean masking, polynomial masking (all of security order d using $n = d+1$ secret shares) and, for completeness, with the original IP masking (of security order $d = 1$ using $n = 4$ secret shares).

Following previous work [1,18,29,35,36] we use the mutual information between a variable and the leakage of all shares of its masked representation as criterion for the comparison. We estimate the mutual information using computer simulations.

We evaluate our improved IP masking for $d = 1$ ($L_1 = 1$, $L_2 = 255$) and for $d = 2$ ($L_1 = 1$, $L_2 = 15$, $L_3 = 233$). Boolean masking uses $d + 1$ shares $(M_1, \ldots, M_d, V)$ where the $M_i \in_R \mathbb{F}_{2^8}$ and V is computed such that $S = M_1 + \ldots + M_d + V$ holds. We evaluate Boolean masking for $d \in \{1, 2, 3\}$. Polynomial masking uses $d + 1$ public coefficients $(\alpha_1, \ldots, \alpha_{d+1})$ with $\alpha_i \in_R \mathbb{F}_{2^8} \setminus \{0\}$ and pairwise distinct, and $d + 1$ shares $(S_1, \ldots, S_{d+1})$ with $S_i = P_S(\alpha_i) \in \mathbb{F}_{2^8}$ [29].

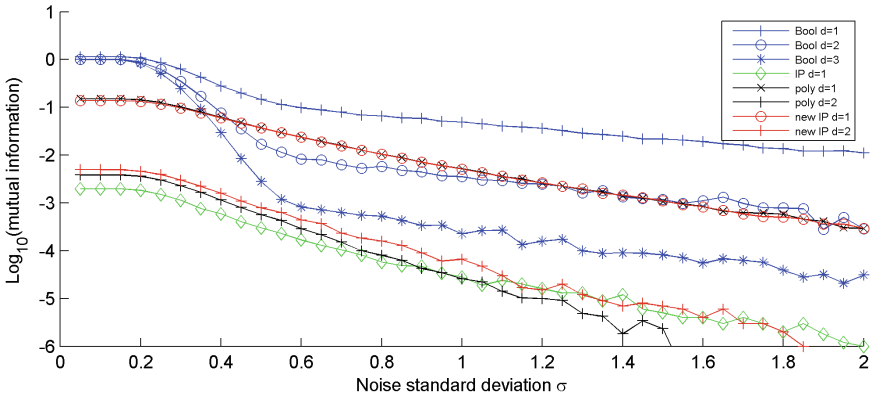

Fig. 1. Mutual information ($\log_{10}$) over increasing noise standard deviation σ for different masking functions

We evaluate polynomial masking for $d = 1$ ($\alpha_1 = 3, \alpha_2 = 7$) and $d = 2$ ($\alpha_1 = 13, \alpha_2 = 240, \alpha_3 = 163$). For the original IP masking we set $d = 1$ and let $R_2 \in_R \mathbb{F}_{2^8}$ and $L_1, L_2 \in_R \mathbb{F}_{2^8} \setminus \{0\}$ such that $S = L_1 \times R_1 + L_2 \times R_2$.

We model the relation between a share and its physical leakage as usual in the literature: each share leaks its Hamming weight, each share leaks independently of all other shares, and each leakage is affected by independent Gaussian noise. In summary, we model the leakage of our improved scheme as

$$\mathrm{Leak}(\boldsymbol{L}, \boldsymbol{R}) = (\mathrm{HW}(R_1) + n_1, \ldots, \mathrm{HW}(R_{d+1}) + n_{d+1}),$$

the leakage of boolean masking as

$$\mathrm{Leak}(M_1, \ldots, M_d, V) = (\mathrm{HW}(M_1) + n_1, \ldots, \mathrm{HW}(M_d) + n_d, \mathrm{HW}(V) + n_{d+1}),$$

the leakage of polynomial masking as

$$\mathrm{Leak}(S_1, \ldots, S_{d+1}) = (\mathrm{HW}(S_1) + n_1, \ldots, \mathrm{HW}(S_{d+1}) + n_{d+1})$$

and the leakage of the original IP masking as

$$\mathrm{Leak}(\boldsymbol{L}, \boldsymbol{R}) = (\mathrm{HW}(L_1) + n_1, \mathrm{HW}(R_1) + n_2, \mathrm{HW}(L_2) + n_3, \mathrm{HW}(R_2) + n_4),$$

where the n_i are independent Gaussian variables with mean zero and standard deviation σ. The mutual information between the secret variable and the leaked information is then $I(S; \mathrm{Leak}(\boldsymbol{L}, \boldsymbol{R}))$, $I(S; \mathrm{Leak}(M_1, \ldots, M_d, V)$, and $I(S; \mathrm{Leak}(S_1, \ldots, S_{d+1}))$ respectively. Recall that the mutual information is directly related to the number of measurements that a Template Attack [6] (worst case attack scenario) requires to achieve a given success probability. Standaert et al. [36] defined the relation via $c \cdot I(\cdot;\cdot)^{-1}$ where the constant c is related to the success probability.

Figure 1 shows plots of the mutual information (in $\log_{10}$ scale) between S and the information leaked by all shares of its masked representation, over increasing noise levels σ, for all masking types considered.

The results are in line with our expectations. Our improved IP masking leaks more information than the original IP masking (in the improved scheme $\boldsymbol{L}$ is public), which illustrates the security versus efficiency tradeoff. But it leaks less than Boolean masking with the same number of shares, which is due to the more complex algebraic relation between the shares and the secret variable. The difference is particularly pronounced for low levels of noise and it seems to increase with increasing security order d. Finally, our improved IP masking and polynomial masking leak comparably. We attribute this to their similar representation of variables in the masked domain.

5 Security Proof in the Probing Model

We start our security analysis with a proof in the so-called probing model introduced by Ishai, Sahai and Wagner [21]. Recall that our masking scheme has the form $(\boldsymbol{L}, \boldsymbol{R})$, where $\boldsymbol{L}$ is public and $\boldsymbol{R}$ is secret and random in $\mathcal{K}^n$ for $n \in \mathbb{N}$ being the security parameter. We will show that our construction is secure against any d probing adversary, where we assume that $n = 2d+1$. That is, an adversary that can learn up to d arbitrary intermediate values computed during the execution of the masked scheme will not learn anything about the underlying secrets. Let in the following denote by $\mathcal{P}$ the set of intermediate values that the adversary is probing.

As a first step we show that our masking function is indeed secure against an $(n-1)$-probing adversary. We use the notation $\mathcal{A}_{n-1}(\texttt{IPMask}_{\boldsymbol{L}}(S))$ to describe that the adversary $\mathcal{A}$ obtains at most $n-1$ shares of the masking.

Lemma 1. *For any two secrets $S, S' \in \mathcal{K}$ and any $(n-1)$-probing adversary $\mathcal{A}$ we have*

$$\mathcal{A}_{n-1}(\boldsymbol{S}) = \mathcal{A}_{n-1}(\boldsymbol{S'}),$$

where $\boldsymbol{L}$ was sampled as specified by the setup algorithm and $\boldsymbol{S} \leftarrow \mathtt{IPMask}_{\boldsymbol{L}}(S)$ and $\boldsymbol{S'} \leftarrow \mathtt{IPMask}_{\boldsymbol{L}}(S')$.

Proof. Notice that in our scheme the vector $\boldsymbol{L}$ is public and hence does not contribute to the security against probing attacks. Further, recall that all $L_i \neq 0$, and hence all values of $\boldsymbol{R}$ contribute to the security of the masking. The proof follows by the fact that given $\boldsymbol{L}$ with all components $\neq 0$ the vector $\boldsymbol{R}$ is a perfect random additive $(n-1)$ out of n secret sharing scheme. More precisely, let $\mathcal{I} \subset [n]$ be the subset of indices for which $\mathcal{A}_{n-1}$ learns R_i, i.e., for all $i \in \mathcal{I}$ we have that $\mathcal{A}_{n-1}$ probes R_i. As $|\mathcal{I}| < n$ there exists at least one $j \in [n]$ such that $j \notin \mathcal{I}$. Hence, for any value $S \in \mathcal{K}$, any choice of $\boldsymbol{L}$ and any R_i such that $i \in \mathcal{I}$ there exists R_j such that $S = \langle \boldsymbol{L}, \boldsymbol{R} \rangle$. □

5.1 Security of Masked Operations

We now show the security of the different operations presented in Section 3. Informally, we will show that any subset of wires $\mathcal{P}$ with size $|\mathcal{P}| \leq d$ is independent of the underlying masked values, i.e., the probes $\mathcal{P}$ given to the adversary

will not help the adversary in breaking the security guarantee of the underlying scheme. To this end, we will prove the security for any masked operation individually and then show that also a combination of such masked operations remains secure if the adversary obtains at most d probes in the entire circuit.

The proof is easy for the masked addition operation: probes at the inputs and outputs directly translate to probes at the underlying masked value. We will show security for the masked squaring and masked multiplication operation. The proof of the masked multiplication is rather tedious since simulating the intermediate values just from the encoded inputs and outputs of the masked operations requires careful bookkeeping.

We will denote by $\mathcal{A}_d(\mathsf{Operation}(\boldsymbol{X}, \boldsymbol{Y}))$ the output of an adversary that probes up to d values in the execution of the operation $\mathsf{Operation}$ when run on masked inputs $\boldsymbol{X}$ and $\boldsymbol{Y}$. Notice that $\mathcal{A}_d$ may also probe the output produced by $\mathsf{Operation}$, i.e., $\boldsymbol{Z} \leftarrow \mathsf{Operation}(\boldsymbol{X}, \boldsymbol{Y})$. As an example let $\mathsf{Operation}$ be the $\mathtt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}(\boldsymbol{Q}, \boldsymbol{R})$ operation on inputs $\boldsymbol{Q}$ and $\boldsymbol{R}$. The adversary may learn up to d of the intermediate values produced during the computation of this algorithm. Moreover, for a vector $\boldsymbol{X}$ and a subset $\mathcal{I} \subset [n]$ we denote by $\boldsymbol{X}_{|\mathcal{I}}$ the set $\{X_i\}_{i \in \mathcal{I}}$. Following Ishai et al. we will say that a masked operation $\mathsf{Operation}$ is secure against d probing attacks if probes on intermediate values produced by the masked operation $\mathsf{Operation}$ can be simulated by just access to the inputs of the operation.

Security of the masked squaring operation. It is simple to show security of the squaring algorithm presented in Algorithm 6 in the probing model, where $d < n$.

Lemma 2. *Let n be the security parameter and let $d < n$, then for any d-probing adversary $\mathcal{A}_d$ and any $R \in \mathcal{K}$ and $\boldsymbol{R} \leftarrow \mathit{IPMask}_{\boldsymbol{L}}(R)$, there exists a subset $\mathcal{I} \subset [n]$ with $|\mathcal{I}| \leq d$ and a simulator $\mathsf{Sim}(\boldsymbol{R}_{|\mathcal{I}})$ such that:*

$$\mathcal{A}_d(\mathit{IPSquare}_{\boldsymbol{L}}(\boldsymbol{R})) \equiv \mathsf{Sim}(\boldsymbol{R}_{|\mathcal{I}}).$$

Proof. We start by a description of how to build the set $\mathcal{I}$, which initially is set to $\mathcal{I} = \{\}$. For each probe of the form $R_i, (R_i)^2$ or $(R_i)^2 \times L_i$ add the index i to $\mathcal{I}$. Since the adversary can make at most d probes, we clearly have $|\mathcal{I}| \leq d$. Given the set $\boldsymbol{R}_{\mathcal{I}}$ and $\boldsymbol{L}$ (which is public and hence the simulator has access to it), it is easy to simulate all probes in $\mathtt{IPSquare}_{\boldsymbol{L}}(\boldsymbol{R})$ in a perfect way and in particular consistent with probes on the real execution of the squaring algorithm. □

Security of masked multiplication. We prove the security of Algorithm 5 in the d-probing model, where $n \geq 2d + 1$.

Lemma 3. *For any $Q, R \in \mathcal{K}$ let $\boldsymbol{Q} \leftarrow \mathit{IPMask}_{\boldsymbol{L}}(Q)$ and $\boldsymbol{R} \leftarrow \mathit{IPMask}_{\boldsymbol{L}}(R)$. Let n be the security parameter and let d be such that $2d < n$, then for any d-probing adversary $\mathcal{A}_d$ that learns at most d probes on intermediate values produced during the masked multiplication $\mathit{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}(\boldsymbol{Q}, \boldsymbol{R})$, there exists a subset $\mathcal{I} \subset [n]$ with $|\mathcal{I}| \leq 2d$ and a simulator $\mathsf{Sim}(\boldsymbol{Q}_{|\mathcal{I}}, \boldsymbol{R}_{|\mathcal{I}})$ such that:*

$$\mathcal{A}_d(\mathit{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}(\boldsymbol{Q}, \boldsymbol{R})) \equiv \mathsf{Sim}(\boldsymbol{Q}_{|\mathcal{I}}, \boldsymbol{R}_{|\mathcal{I}}).$$

Proof. To simplify the analysis we assume that $\boldsymbol{L}$ is given in its entirety a-priori to the adversary. We show that the entire distribution of the multiplication algorithm can be simulated by having access to at most $2d$ shares of $\boldsymbol{Q}$ and $\boldsymbol{R}$, respectively. Since by Lemma 1 seeing $2d$ shares of $\boldsymbol{Q}$ and $\boldsymbol{R}$ respectively, is independent of the masked secret this proves the security of the masked multiplication operation. The set that keeps track of what values are revealed of $\boldsymbol{Q}, \boldsymbol{R}$ is called $\mathcal{I}$. Moreover, we keep track of two sets $\mathcal{T}$ and $\mathcal{U}$ that are needed to keep the simulation consistent. $\mathcal{T}$ keeps tuples $(i,j) \subset [n] \times [n]$ of pairs that correspond to the values of $A_{i,j}$ which are revealed during a probing attack, while $\mathcal{U}$ represents the values (i,j) of $B_{i,j}$ that are revealed by the attack. To simplify our analysis, we will be rather generous to the adversary and usually give him much more values than what are revealed during the actual probe of the particular intermediate value. Below we describe how to build the set $\mathcal{I}$.

A. Building the sets $\mathcal{I},\mathcal{U},\mathcal{T}$***:*** In this step we initialize the sets that later are needed for the simulation.

1. Initially, we set the sets $\mathcal{I},\mathcal{U},\mathcal{T}$ to the empty set $\{\}$.
2. *Probes when* Δ_i *is computed, i.e., probes of the form* $A_{1,i}, \ldots, A_{n,i}$ *or probes of the form* $\Delta_{i-1} + \sum_{1 \le j < n} L_{j,i} \times A_{j,i}$*:* Add the index $(1,i), \ldots, (n,i)$ into the set $\mathcal{T}$.
3. *Probes of the form* $B_{i,j}$ *or sums of the form* $\sum_j L_{i,j} \times B_{i,j}$*:* We distinguish two cases:
   (a) For $i > 1$: Add the index $(i,1), \ldots, (i,n)$ to the set $\mathcal{U}$.
   (b) For $i = 1$: Add the index (i,j) to the set $\mathcal{U}$.
   The above two cases capture the fact that for row $i = 1$ of the matrix $\hat{\boldsymbol{B}}$ we do not compute the sum of values $B_{1,j}$, i.e., of the first row of the matrix $\hat{\boldsymbol{B}}$. Additionally, for each such (i,j) that has been added to $\mathcal{U}$: if (i,j) is in $\mathcal{T}$, then add i,j to $\mathcal{I}$.
4. *Probes of the form* Q_i, R_j *and* $Q_i \times R_j$*:* For probes of the form Q_i resp. R_j add i resp. j to $\mathcal{I}$. For a probe of the form $Q_i \times R_j$ add the indices i,j to $\mathcal{I}$.

Given the above description of the the sets $\mathcal{I}, \mathcal{T}$ and $\mathcal{U}$, we can now define the simulator $\mathsf{Sim}_{\mathcal{I},\mathcal{T},\mathcal{U}}(\boldsymbol{R}_{|\mathcal{I}}, \boldsymbol{Q}_{|\mathcal{I}})$.

B. Sampling variables independently of probes: We start by sampling some of the values a-priori before we answer the actual probing queries. We will later take care that all probes are answered consistently with the values sampled in this initial step.

1. Choose $\beta_2, \ldots, \beta_n$ uniformly at random. Notice that this allows to compute all the potential values that appear in the sum when computing the value b – including the value b.
2. Sample $\Delta_1, \ldots, \Delta_{n-1}$ uniformly at random and set $\Delta_0 = \Delta_n = 0$.

C. Simulating the probes: We next show how to answer the probing queries of the adversary given all values $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$, and the values Δ_i, β_j sampled in Step B.

1. *Probes of the form $\beta_2, \ldots, \beta_n$ and sums thereof:* These values have been fixed in Step B1 and hence probes on these values can easily be answered from the above sampled values.
2. *Probes on the sampling of $A_{i,j}$:* Notice that this involves the individual values $A_{i,j}$ as well as sub-sums of values in the columns with the appropriate values Δ_i. If $(1,i), \ldots, (n,i)$ are in $\mathcal{T}$ then sample $A_{1,i}, \ldots, A_{n,i}$ uniformly at random such that $\sum_j L_{j,i} \times A_{j,i} = \Delta_i + \Delta_{i-1}$. Given the above sampled values and the values Δ_i sampled in Step B2, we can answer any probe of the adversary.
3. *Probes of the form Q_i, R_j and $Q_i \times R_j$:* Given access to $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$ we can easily simulate the probes in a consistent way.
4. *Probes of $B_{i,j}$, or when $i > 1$ of sums thereof:* To answer these probes, we sample the values of $B_{i,j}$ in the following way:
   (a) If (i,j) is not in $\mathcal{U}$ then leave the values $B_{i,j}$ un-assigned.
   (b) If (i,j) is in $\mathcal{U}$ and (i,j) is in $\mathcal{T}$ then compute $B_{i,j} = Q_i \times R_j + A_{i,j}$. Notice that this is possible since the relevant values of Q_i and R_j are given in $\{Q_i\}_{i \in \mathcal{I}}$ and $\{R_i\}_{i \in \mathcal{I}}$ and $A_{i,j}$ has been assigned in Step C2.
   (c) If (i,j) is in $\mathcal{U}$, but (i,j) is not in $\mathcal{T}$, then we sample $B_{i,j}$ uniformly at random subject to the constraint that $\beta_i = \sum_j L_{i,j} \times B_{i,j}$. Notice that the later requirement only is needed for $i > 1$. The value $B_{1,j}$ is chosen uniformly at random.

We will show below that (1) the simulation has the same distribution as the real execution of the masked multiplication operation (second claim below), and (2) we argue that the size of the set $\mathcal{I}$ has always cardinality $|\mathcal{I}| \leq 2d$ (first claim below). Putting these two claims together proves the lemma.

In the following analysis we denote by u the number of probes corresponding to Step A2, by v the number of probes corresponding to Step A3 and by w the number of probes corresponding to Step A4.

Claim. Let $n \in \mathbb{N}$ be the security parameters and d be the number of probes such that $2d < n$. Then, $|\mathcal{I}| \leq 2d$.

Proof. Observe that the simulator adds elements to $\mathcal{I}$ only in Step A3 and Step A4. As each probe in Step A4 leads to adding at most two elements to $\mathcal{I}$ and $w \leq d$, this directly implies $|\mathcal{I}| \leq 2d$ if probes appear only in Step A4. It remains to analyze the number of elements we add to $\mathcal{I}$ for each probe done in Step A3. The analysis for Step A3 is a little more involved as the number of elements added to $\mathcal{I}$ depends on both u (number of probes in Step A2) and v (number of probes in Step A3). Recall that for each probe of Step A2 that is within the i-th column we add all indices $(1,i), \ldots, (n,i)$ to $\mathcal{T}$ that correspond to the elements in the i-th column of the matrix $\hat{\boldsymbol{A}}$, while for each probe in Step A3 we add all indices $(i,1), \ldots, (i,n)$ to $\mathcal{U}$ that correspond to the i-th row of the matrix $\hat{\boldsymbol{B}}$ (except for the first row, but this does not matter for the rest of the analysis). Furthermore, recall that each $B_{i,j}$ is computed from $A_{i,j}$. Depending on the values added to $\mathcal{T}$ and $\mathcal{U}$, in Step A3 we will add all i,j to $\mathcal{I}$ where (i,j) is an "intersection" between the columns and rows mentioned above.

Unfortunately, it is easy to see that for u columns and $v = d - u - w$ rows we may have more than $2d$ intersections.[1] The good news is, however, that many elements will be added multiple times to $\mathcal{I}$, hence not increasing the size of $\mathcal{I}$.

More precisely, for a query in the i-th row from Step A3, we add the index i into the set $\mathcal{I}$. Additionally, for each such query we add index of the column at which we have an intersection from a query in the j-th column. The main observation is that for each row the indices added by the intersections with the columns are the same. In other words, for each i the tuples (i, j) have the same second component, and hence we add at most $u + v \leq d$ indices to $\mathcal{I}$ in Step A3. Combining it with the probes from Step A4, we add $u + v + 2w$ elements to $\mathcal{I}$. Since $u + v + w \leq d$, we get that $|\mathcal{I}| \leq 2d$, which proves the claim. □

The next claim shows that the sampling of the simulator produces the same view as a d-probing adversary obtains in a real attack against the masked multiplication operation.

Claim. For any $Q, R \in \mathcal{K}$ let $\boldsymbol{Q} \leftarrow \mathtt{IPMask}_{\boldsymbol{L}}(Q)$ and $\boldsymbol{R} \leftarrow \mathtt{IPMask}_{\boldsymbol{L}}(R)$, we have:

$$\mathcal{A}_d(\mathtt{IPMult}_{\boldsymbol{L},\hat{\boldsymbol{L}}}(\boldsymbol{Q}, \boldsymbol{R})) \equiv \mathsf{Sim}(\boldsymbol{Q}_{|\mathcal{I}}, \boldsymbol{R}_{|\mathcal{I}}),$$

with parameters defined as in the statement of the lemma.

Proof. By the last claim we have $|\mathcal{I}| \leq 2d$. We compare the way in which the probes are answered by the simulator in Steps B and C with the real attack against the execution of the masked multiplication.

Step B: The joint distribution of values sampled at this step in the simulation is identically distributed with the real experiment even given all these values to the adversary. This is easy to see for the β_i values as they are just sums of random values $A_{i,j}$. Moreover, for the Δ_j values we observe that the first row is never used in computing β_i (since $i > 1$), and hence all Δ_j for $j < n$ can be chosen uniformly at random (they can essentially be made consistent with the view of the adversary by choosing $A_{1,j}$ appropriately. The value Δ_n is fixed to 0 as we require $\langle \hat{\boldsymbol{A}}, \hat{\boldsymbol{L}} \rangle = 0$.

Step C2: In Step B2 we sampled $\Delta_1, \ldots \Delta_{n-1}$ uniformly at random and in Step C2 we sample each column $(A_{1,i}, \ldots, A_{n,i})$ of the matrix $\hat{\boldsymbol{A}}$ uniformly at random such that $\Delta_i + \Delta_{i-1} = \sum_{j \in [n]} L_{j,i} \times A_{j,i}$. This implies that all $A_{i,j}$ for $j \in [n]$ are chosen uniformly at random[2]. For the last column of the matrix we require that $\Delta_n + \Delta_{n-1} = \Delta_{n-1} = \sum_{j \in [n]} L_{j,n} \times A_{j,n}$, which guarantees that $\langle \hat{\boldsymbol{A}}, \hat{\boldsymbol{L}} \rangle = 0$ as required by the protocol. It remains to show that the choice of the $A_{i,j}$ values produced by the simulator is consistent with the simulator's choice of the values β_i.

To this end observe that after a probing attack at most $u \leq d$ columns of the matrix $\hat{\boldsymbol{A}}$ have been assigned, i.e., the values in $n - d$ columns remain

[1] For instance, suppose that $u = d - 3$ and $v = 3$, then we add $3d - 9$ elements to the set $\mathcal{I}$.

[2] At this step we also require that $L_{i,j} \neq 0$ as required by our scheme.

un-assigned. As $\beta_i = a + \sum_j A_{i,j}$ for some fixed value $a = Q_i \times L_i \times R$ the value β_i is distributed uniformly at random even given all values Δ_i and all values $A_{i,j}$ that haven been assigned previously. Notice that this is the case since (a) there is at least one column in the matrix $\hat{\boldsymbol{A}}$ that has not been assigned yet, and (b) the scheme never computes $\sum_j A_{1,j}$ as β_1 is not needed for the computation. More precisely, let j^* be the column that is unassigned after the assignment of the values above, then each β_i is perfectly hidden by A_{i,j^*} and we can choose A_{1,j^*} in such a way that the column is consistent with Δ_{j^*}. This concludes the analysis of Step C2.

Step C3: To answer the queries on Q_i, R_j and $Q_i \times R_j$ we use the values given in $\{Q_i\}_{i\in\mathcal{I}}$ and $\{R_i\}_{i\in\mathcal{I}}$. To argue that this gives us the right distribution (independent of Q and R), we first recall that by the last claim $|\mathcal{I}| \leq 2d$. Notice that these values will always be consistent with the previously assigned values since each β_i is still blinded by at least one un-assigned value A_{i,j^*}. Moreover, probes of this form (by itself) do not reveal any additional information about $\hat{\boldsymbol{A}}$.[3]

Step C4: We can ignore the values sampled in Step C4a as these values remain un-assigned, i.e., they are never directly probed nor are they used in the computation of other probes. The values sampled in Step C4b can be computed from the values that have been assigned previously (since (i, j) was in $\mathcal{U}$ and $\mathcal{T}$ which implies that $i, j \in \mathcal{I}$), and, hence will not affect the joint distribution. Notice that in this step we will always only fix a subset of the $B_{i,j}$ elements in the i-th row, because there are at most $u \leq d$ probes in Step A2 (which lead to adding tuples (i, j) to $\mathcal{T}$). The remaining values for the i-th row are chosen as defined in Step C4c. That is, in the simulation we choose these values uniformly at random such that $\beta_i = \sum_j L_{i,j} \times B_{i,j}$ taking into account the previously assigned values for $B_{i,j}$ from Step C4b. By the requirement given in Step C4c our choice of $B_{i,j}$ is consistent with the choice of β_i.

It remains to argue why the simulator's choice is also consistent with the matrix $\hat{\boldsymbol{A}}$ as sampled in Step C2 and with $\Delta_1, \ldots, \Delta_n$. To this end, observe that the simulator only samples $B_{i,j}$ according to Step C4c if $A_{i,j}$ has not been revealed previously (i.e., it was in a column of $\hat{\boldsymbol{A}}$ that has never been probed). Hence, indeed $B_{i,j}$ is uniformly distributed (since it is blinded by the random and unknown value $A_{i,j}$). Finally, it remains to argue that the choice of $B_{i,j}$ is also consistent with the choice of Δ_j from Step B. Recall that for values $B_{i,j}$ for which $(i, j) \in \mathcal{U}$, but not in $\mathcal{T}$ this implies that the j-th row has not been queried. Moreover, we know since $v \leq d$ there are at least $n - v > d$ rows that have not been probed. Hence, there exists some $B_{k,j}$ and $A_{k,j}$ that have not been assigned during the experiment (i.e., they belong to Step C4a). Such values $A_{k,j}$ were never used in the experiment and can always be chosen such that the total sum is consistent with Δ_j.

The above description concludes the proof. □

[3] They can, however, reveal information about $B_{i,j}$ as we will see in the next step.

Putting together the above two claims we obtain the statement of the lemma. □

This completes the analysis of the security of individual masked operations. In the next section we briefly argue about security of masked composed algorithms.

5.2 Security of General Masked Computation

In Section 5.1 we showed that an adversary that probes up to d intermediate values in the computation of the basic masked operation will not be able to learn anything about the underlying sensitive information. We are now interested in what happens to the security when multiple of such operations are combined to carry out some more complicated computation such as the AES encryption algorithm. In other words, we let the adversary learn d intermediate values of the computation carried out by the masked AES algorithm (or any other masked algorithm). Notice that this in particular requires that, e.g., learning d_1 values in a masked multiplication cannot be exploited together with d_2 values learnt from a consecutive masked squaring algorithm as long as $d_1 + d_2 \leq d$.

Similar to earlier work [21], we only provide an informal analysis of d-probing security of composed masked operations. To this end, observe that both in Lemma 2 and Lemma 3 the simulation only depends on at most $2d$ elements of the outputs of the masked operation. As an example consider the masked multiplication that outputs the vector $\boldsymbol{T}$ and assume that $\boldsymbol{T}$ is input for a squaring algorithm. If the adversary probes d_1 intermediate values in the multiplication operation, then it is easy to see that the simulation described in Lemma 3 depends on at most $2d_1$ shares of $\boldsymbol{T}$. Moreover, the masked multiplication operation guarantees that even given $\boldsymbol{Q}$ and $\boldsymbol{R}$ entirely, the output $\boldsymbol{T}$ is a uniformly and independently chosen maskings of $Q \times R$. This means that for the simulation of the adjacent squaring algorithm the simulator starts with a masking of which $2d_1$ shares are already known. Since according to Lemma 2 the simulator requires d_2 elements of its inputs to simulate the probes in the squaring operation, the simulator will learn additionally d_2 elements of $\boldsymbol{T}$. Since $2d_1 + d_2 < n$ this shows security of the simple composed circuit consisting of a multiplication followed by a masked squaring operation. The above argument can easily be extended to arbitrary complicated masked algorithms consisting of many masked operations.

Another difficulty occurs when the adversary can run the masked algorithm multiple times and in each execution he may observe d intermediate values. For instance, one may think of a masked AES algorithm running with a masked key K. Notice that this setting is different from the setting of composed masked computation described above since now the adversary can observe qd intermediate values, where q is the number of executions that the masked algorithm is run. As in earlier work [7,21] the problem of a continuous probing adversary, i.e., an adversary that learns up to d intermediate values in each execution of the masked AES algorithm, can be addressed by a key refresh algorithm. If $(\boldsymbol{L}, \boldsymbol{R})$ denotes the masking of a key byte of the AES, then the masking of this key byte can be refreshed by running the Algorithm 3 n times consecutively. In other words, the

key refresh algorithm takes as input $\boldsymbol{R}_0 := \boldsymbol{R}$ and for $i = 1, \ldots n$ proceeds as follows: Compute $\boldsymbol{R}_i = \texttt{IPRefresh}_{\boldsymbol{L}}(\boldsymbol{R}_{i-1})$ and output $\boldsymbol{R}' = \boldsymbol{R}_n$. Clearly, since we execute $\texttt{IPRefresh}_{\boldsymbol{L}}$ n times and the adversary can probe only $d < n$ intermediate values there must exist at least one execution of $\texttt{IPRefresh}_{\boldsymbol{L}}(\boldsymbol{R}_{i-1})$ that does not leak at all. Hence the mask of $\boldsymbol{R}$ is completely refreshed. This enables us to transform our result to a continuous probing adversary without any additional loss, when in each execution the adversary can learn up to d values (where $2d < n$).

We notice that the fact that $\texttt{IPRefresh}_{\boldsymbol{L}}$ is repeated multiple times to refresh the masking of the key is not only an artefact of the security proof. In fact, it is easy to show that for natural implementations of $\texttt{IPRefresh}_{\boldsymbol{L}}$ the scheme becomes insecure. To illustrate this, we present a simple attack against a more efficient key refresh algorithm that executes $\texttt{IPRefresh}_{\boldsymbol{L}}$ only a single time. For simplicity, let us assume that $\boldsymbol{L} = (1, \ldots, 1)$, i.e., the vector is the all-1 vector. Notice that in this case the inner product masking function is identical to the Boolean masking used, e.g., in [33]. Let us assume that $\boldsymbol{A}$ is sampled in $\texttt{IPRefresh}_{\boldsymbol{L}}$ in the following way:

1. Let $t_0 = 0$. For $i = 1, \ldots, n-1$ repeat the following:
   (a) Sample A_i uniformly at random in $\mathcal{K}$
   (b) Compute $t_i = t_{i-1} + A_i$.
2. Set $A_n = t_i$ and output $\boldsymbol{A} = (A_1, \ldots, A_n)$.

Consider a masked implementation of the AES that at the end of the execution refreshes its key shares by applying a single execution of the $\texttt{IPRefresh}_{\boldsymbol{L}}$ algorithm. We now describe an attack that allows to recover the key with only 2 probes in each execution of the masked AES implementation. Notice that we consider the *full probing model* [7], where the adversary can move its probes between consecutive rounds of execution. Our attack does not apply in the restricted probing model [7]. We denote by $\boldsymbol{K}^i$ the masking of the key k at the beginning of the i-th round, i.e., for all i we have $\langle \boldsymbol{L}, \boldsymbol{K}^i \rangle = k$. We have $\boldsymbol{K}^0$ being the initially shared key. Moreover, we denote by $\boldsymbol{A}^i$ the vector that is used in the i-th execution of the masked AES implementation for refreshing, i.e., $\boldsymbol{K}^i = \boldsymbol{K}^{i-1} + \boldsymbol{A}^i$.

1. *First execution of the masked AES:* Probe K_1^0 and A_1^0. Notice that this allows us to compute K_1^1.
2. *In the i-th execution of the masked AES:* Probe K_i^i and t_{i-1}^i.

We will now describe how to compute k from the above described probes. Suppose at the beginning of the i-th round (i.e., before carrying out the probes in this round) the adversary knows $\sum_{j=1}^{i-1} K_j^{i-1}$. We show how with the probes described above he can compute $\sum_{j=1}^{i} K_j^i$. To this end, notice that:

$$\sum_{j=1}^{i} K_j^i = \sum_{j=1}^{i-1} K_j^{i-1} + t_{i-1}^i + K_i^i,$$

where the second and the third term the adversary knows from the probes in the i-th round and the first term he knows by assumption. Hence, by induction it is easy to argue that the above attack allows to recover the secret key k.

6 Performance Evaluation

We have applied our masking scheme to protect a software implementation of AES-128 encryption. For illustrative purposes we have opted to develop two implementations employing $n = 2$ and $n = 3$ shares, respectively. This choice not only enables a better comparison with other higher-order schemes, but also allows us to gain insight into how the performance scales with an increasing number of shares.

Our target platform is a legacy AVR ATMega163 microcontroller. This device has an 8-bit architecture and offers 32 general purpose registers, 1 024 bytes of internal SRAM and 16 KBytes of Flash memory. Our implementation is aimed for speed. To this end, we have written all operations in assembly code and made use of lookup tables whenever possible.

The lowest implementation layer corresponds to arithmetic in the field $\mathbb{F}_{2^8}$. Field addition is very efficient, as it can be performed in one clock cycle via the native XOR instruction. Field multiplication on the other hand is not part of the AVR instruction set, and we opt to implement it using `log` and `alog` tables [38]. Because this method contains a conditional statement, i.e. check if any of the operands equals zero, realizing it with a constant flow of instructions requires in our implementation 22 cycles. Field squaring - as well as raisings to the power of four and sixteen - are implemented by means of lookup tables. Our platform does not have internal support for generating random numbers, as opposed to e.g. JavaCard smart cards. For the sake of completeness and testing, random numbers are provided externally and stored in memory.

The public parameters $\boldsymbol{L}$ and $\hat{\boldsymbol{L}}$ are initialized at setup time and kept constant for each execution of the cipher. Consequently, they are hardcoded in Flash memory. Note that for $n = 2$, there exist 2^8 possible vectors $\boldsymbol{A}$ orthogonal to $\boldsymbol{L}$ satisfying $\langle \boldsymbol{A}, \boldsymbol{L} \rangle = 0$. These vectors can be as well precomputed during initialization and stored in Flash memory as a look-up table T satisfying $T(A_2) = A_1 = L_2 \times A_2$. During $\texttt{IPRefresh}_{\boldsymbol{L}}$, a value A_2 is picked at random and the corresponding value A_1 is looked up as $T(A_2)$. This allows to improve the efficiency of $\texttt{IPRefresh}_{\boldsymbol{L}}$ at the cost of storing 256 bytes in Flash memory.

The main difficulty of applying our masking scheme (and any other) to AES consists in efficiently masking its nonlinear part, i.e. the `SubBytes` transformation. In software contexts it is common to implement this transformation by means of a lookup table. While there exist techniques in the literature to protect table lookups at higher-order, e.g. [7], these are rather costly in terms of performance and storage. Alternatively, one can compute the full `SubBytes` step by using the following equation over $\mathbb{F}_{2^8}$ for a given input state byte X:

$$\begin{aligned}\texttt{SubBytes}[X] = \{05\} \times X^{254} + \{09\} \times X^{253} + \{f9\} \times X^{251} + \{25\} \times X^{247} + \\ \{f4\} \times X^{239} + X^{223} + \{b5\} \times X^{191} + \{8f\} \times X^{127} + \{63\}.\end{aligned}$$

This equation involves multiple field operations (particularly costly multiplications) in order to calculate the different powers of X. It is therefore not very suitable if one aims for an efficient implementation. We therefore follow a different path as already done in [1], i.e. we carry out both steps of the `SubBytes` transformation (inverse and affine transformation) separately. Specifically, we first compute the field inverse by using the following power function:

$$\text{Inverse}[X] = X^{254} = \left((X^2 \times X)^4 \times (X^2 \times X)\right)^{16} \times (X^2 \times X)^4 \times X^2.$$

As discussed in [33], this equation can be efficiently carried out with only 4 multiplications and 7 squarings. For the affine transformation (linear only in $\mathbb{F}_2$) we employ the following equation over $\mathbb{F}_{2^8}$:

$$\begin{aligned}\text{AffTrans}[X] = \{05\} \times X^{128} + \{09\} \times X^{64} + \{f9\} \times X^{32} + \{25\} \times X^{16} + \\ \{f4\} \times X^8 + \{01\} \times X^4 + \{b5\} \times X^2 + \{8f\} \times X + \{63\},\end{aligned}$$

requiring 7 squarings, 8 additions, and 7 multiplications with a constant.

The `MixCol` transformation operates on the AES state column-by-column. In particular, each of the bytes in the $0 \leq j \leq 3$ columns is replaced as:

$$\begin{aligned}s'_{0,j} &= \{02\} \times s_{0,j} + \{03\} \times s_{1,j} + s_{2,j} + s_{3,j} \\ s'_{1,j} &= s_{0,j} + \{02\} \times s_{1,j} + \{03\} \times s_{2,j} + s_{3,j} \\ s'_{2,j} &= s_{0,j} + s_{1,j} \times \{02\} + s_{2,j} + \{03\} \times s_{3,j} \\ s'_{3,j} &= \{03\} \times s_{0,j} + s_{1,j} + s_{2,j} + \{02\} \times s_{3,j}.\end{aligned}$$

From these equations it follows that this step can be implemented using a total of 12 masked additions and 8 masked multiplications by a constant, for each column. In [10], the authors of AES suggest a more efficient way to compute the `MixCol` step by using the so-called `xtime` tables. Such technique takes advantage of the fact that field addition is more efficient than field multiplication in general purpose processors. Due to this, they suggest an alternative approach that requires 15 additions and 4 multiplications by 02, which can be simply performed as table lookups. We employ this technique to compute the `MixCol` transformation.

The performance of our protected AES-128 implementation is given in Table 2 for $n = 2$ and $n = 3$ secret shares. We have also implemented Boolean masking as proposed in [33] with the same number of secret shares. In order to enable a fair comparison, all implementations are developed on the same platform and follow the same optimization strategy. Finally, we also provide the results given in [1] for the original IP masking using $n = 4$ secret shares. Recall that the original IP masking was developed to provide security order $d = 1$ for $n = 4$, the same as our new scheme and Boolean masking for $n = 3$.

The performance results presented in Table 2 are given in clock cycles. The rightmost column shows the execution time of a full AES-128 encryption including key schedule, while the other columns depict the performances achieved for each AES building block.

Table 2. Performance evaluation (in clock cycles) of protected AES implementations. This work and Boolean with $n = 2$ secret shares (top); this work and Boolean with $n = 3$ secret shares (middle); original IP masking with $n = 4$ secret shares (bottom).

		`AddKey`	`SubBytes`	`ShiftRows`	`MixCol`	`NextSubKey`	**Full AES**
$n = 2$	*this work*	364	28 810	100	465	7 416	373 544
	Boolean	364	7 850	100	465	2 175	110 569
$n = 3$	*this work*	476	63 354	150	678	16 148	812 303
	Boolean	476	17 034	150	678	4 568	230 221
$n = 4$	original IP	8 796	117 760	200	27 468	44 437	1 912 000

The improvement with respect to the original IP masking is clear from the results. The overall execution time is reduced by more than a factor 2. This efficiency gain is due to our tweaks in the type of masking leading to an improved construction. In fact, almost all newly proposed operations in the masked domain involve significantly less field operations than in [1]. On the implementation side, this reduction in complexity enables a better usage of the register file, i.e. the number of memory accesses to load/store intermediate results can be significantly reduced.

When compared to Boolean masking for the same security level, our scheme still performs slower. The difference is not entirely due to the operations in the masked domain, but rather to the way the AES affine transformation is defined. Because this operation is linear in $\mathbb{F}_2$, protecting it with Boolean masking requires only a matrix multiplication (table lookup) followed by an XOR operation. In contrast, and due to the higher algebraic complexity of the inner product construction, our implementation needs to compute the more complex formula defined over $\mathbb{F}_{2^8}$. Despite this limitation, our results manage to bridge the gap from approximately a factor 10 to a factor 4. This result makes our proposal an interesting alternative to secure implementations at higher orders.

7 Discussion

In this section we discuss further relevant properties of our improved IP masking scheme and touch on ideas for future work.

Similarities and Differences with Polynomial Masking. A direct consequence of our tweaks is that some characteristics of our construction become closer to those of polynomial masking. In particular, variables in both schemes are encoded using $2n$ shares (L_i, R_i) and (α_i, S_i) respectively, with n shares being public and n shares being secret. The decoding sequences of a masked variable S follow the same pattern of operations, e.g. $S = \langle \boldsymbol{L}, \boldsymbol{R} \rangle = \sum_i L_i \times R_i$ and $S = \sum_i \beta_i \times S_i$. We note, however, that the latter is not a consequence of our

simplifications. The same sequence was already used in [1]. In contrast, a direct effect of our tweaks is that the information leakage of encoded secrets using the mutual information as figure of merit becomes comparable.

Despite these high-level similarities, the low-level constructions proposed in this work to carry out operations in the masked domain are different from those proposed in [18,29]. The only exception is the addition algorithm, in which the secret shares R_i (S_i, respectively) of the input operands are added element-wise to obtain the secret shares of the output. We note that this is also the same for other constructions based on e.g. Boolean masking. The most notable differences can be found in the steps followed to compute the product of two masked variables. In addition, the asymptotic complexity of our multiplication algorithm is $\mathcal{O}(n^2)$ rather than $\mathcal{O}(n^3)$ as presented in [18,29] or $\tilde{\mathcal{O}}(n^2)$ as described in [8]. Asymptotic improvements are possible using more efficient protocols from multiparty computation – in particular, techniques from multiparty computation using packed secret sharing [14,20], which results overall in quasi-linear complexity (for large and parallelizable computation).

Finally, we recall that the very nature of both approaches is different. Polynomial masking employs secure multi-party computation techniques [2] and Shamir's secret-sharing [34], while our construction is inspired by work on leakage resilient cryptography [12]. This difference is for instance prominently reflected in the procedures to mask a variable. In polynomial masking the secret shares are obtained by polynomial evaluation $S_i = P_S(\alpha_i)$ of the public shares α_i, which is different from the procedure described in $\texttt{IPSetup}_n$. Another difference is that the public parameters α_i of polynomial masking must be both *distinct* and nonzero, while for IP masking only the latter requirement applies, i.e. several L_i can have the same value.

Bounded Leakage Model. We notice that we do not prove the security of our construction in the bounded independent leakage model as done in the work of Balasch *et al.* [1]. Instead, the goal of the current work is to develop an *efficient* higher-order masking scheme that exhibits higher algebraic complexity than Boolean masking and prove its security in the ISW probing model. Note that it is still possible to provide a scheme secure in the independent leakage model even if the vector $\boldsymbol{L}$ is public but random. The technical reason for this is that the inner product is a strong extractor, i.e. security holds *even* if one part is revealed completely.

The only requirement we need is that the leakage functions are chosen a-priori and independently of $\boldsymbol{L}$, which allows us to rely on the fact that the inner product function over finite fields is a strong extractor [31].[4] While this may slightly improve the bounds, for small field size, we would still require a large number of shares which may be unrealistic for practical settings. In addition, we would have to make slight changes to the construction, e.g. to our optimized

[4] We notice that a similar non-adaptive leakage model was considered in [13,39]. The attack presented in [13] against [39] would not apply in our case since masking schemes are randomized while the stream cipher construction of [39] is deterministic.

squaring algorithm, to prevent simultaneous access to the two halves $(\boldsymbol{L}, \boldsymbol{R})$ of the IP encoding (which becomes insecure under bounded independent leakages).

Resistance Against Glitches. The original scheme in [1] achieves provable security in the presence of glitches *only* when the security parameter n is large. In fact, the proof for glitch resistance follows directly from security in the bounded independent leakage model. However, the construction in this work is proven secure in the probing model, which *does not* automatically imply glitch resistance. Hence we do not claim that our construction is provable secure in the presence of glitches, in contrast to e.g. the polynomial masking scheme by Prouff and Roche [29] and the Threshold Implementation scheme by Nikova et al. [3,26].

Future Work. An interesting question for future work is if the ideas from [9] can be applied in order to gain a factor 2, i.e. it may be feasible to achieve security against d probes when $d = n - 1$ in the restricted model.

Acknowledgments. Benedikt Gierlichs is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (FWO). This work has been funded in parts the Research Council KU Leuven: GOA TENSE (GOA/11/007). Sebastian Faust received funding from the Marie Curie IEF/FP7 project GAPS, grant number: 626467.

We thank Stefan Dziembowski for many useful discussions on the inner product encoding and the anonymous reviewers of Eurocrypt 2015 for helpful feedback that helped to improve the presentation of our work.

References

1. Balasch, J., Faust, S., Gierlichs, B., Verbauwhede, I.: Theory and Practice of a Leakage Resilient Masking Scheme. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 758–775. Springer, Heidelberg (2012)
2. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Symposium on Theory of Computing, STOC 1988, pp. 1–10. ACM (1988)
3. Bilgin, B., Gierlichs, B., Nikova, S., Nikov, V., Rijmen, V.: Higher-Order Threshold Implementations. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, Part II. LNCS, vol. 8874, pp. 326–343. Springer, Heidelberg (2014)
4. Carlet, C., Goubin, L., Prouff, E., Quisquater, M., Rivain, M.: Higher-Order Masking Schemes for S-Boxes. In: Canteaut, A. (ed.) FSE 2012. LNCS, vol. 7549, pp. 366–384. Springer, Heidelberg (2012)
5. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer, Heidelberg (1999)
6. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Kaliski Jr., B.S., Koç, Ç.K., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 13–28. Springer, Heidelberg (2002)
7. Coron, J.-S.: Higher Order Masking of Look-Up Tables. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 441–458. Springer, Heidelberg (2014)

8. Coron, J.-S., Prouff, E., Roche, T.: On the Use of Shamir's Secret Sharing against Side-Channel Analysis. In: Mangard, S. (ed.) CARDIS 2012. LNCS, vol. 7771, pp. 77–90. Springer, Heidelberg (2013)
9. Coron, J.-S., Prouff, E., Rivain, M., Roche, T.: Higher-Order Side Channel Security and Mask Refreshing. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 410–424. Springer, Heidelberg (2014)
10. Daemen, J., Rijmen, V.: The Design of Rijndael: AES - The Advanced Encryption Standard. Information Security and Cryptography. Springer (2002)
11. Duc, A., Dziembowski, S., Faust, S.: Unifying Leakage Models: From Probing Attacks to Noisy Leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Heidelberg (2014)
12. Dziembowski, S., Faust, S.: Leakage-Resilient Circuits without Computational Assumptions. In: Cramer, R. (ed.) TCC 2012. LNCS, vol. 7194, pp. 230–247. Springer, Heidelberg (2012)
13. Faust, S., Pietrzak, K., Schipper, J.: Practical Leakage-Resilient Symmetric Cryptography. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 213–232. Springer, Heidelberg (2012)
14. Franklin, M.K., Yung, M.: Communication complexity of secure computation (extended abstract). In: Kosaraju, S.R., Fellows, M., Wigderson, A., Ellis, J.A. (eds.) Proceedings of the 24th Annual ACM Symposium on Theory of Computing, 1992, Victoria, British Columbia, Canada, May 4–6, pp. 699–710. ACM (1992)
15. Fumaroli, G., Martinelli, A., Prouff, E., Rivain, M.: Affine Masking against Higher-Order Side Channel Analysis. In: Biryukov, A., Gong, G., Stinson, D.R. (eds.) SAC 2010. LNCS, vol. 6544, pp. 262–280. Springer, Heidelberg (2011)
16. Gandolfi, K., Mourtel, C., Olivier, F.: Electromagnetic Analysis: Concrete Results. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Heidelberg (2001)
17. Genelle, L., Prouff, E., Quisquater, M.: Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 240–255. Springer, Heidelberg (2011)
18. Goubin, L., Martinelli, A.: Protecting AES with Shamir's Secret Sharing Scheme. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 79–94. Springer, Heidelberg (2011)
19. Goubin, L., Patarin, J.: DES and Differential Power Analysis. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 158–172. Springer, Heidelberg (1999)
20. Grosso, V., Standaert, F., Faust, S.: Masking vs. multiparty computation: how large is the gap for AES? J. Cryptographic. Engineering **4**(1), 47–57 (2014)
21. Ishai, Y., Sahai, A., Wagner, D.: Private Circuits: Securing Hardware against Probing Attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Heidelberg (2003)
22. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
23. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 388–397. Springer, Heidelberg (1999)
24. Mangard, S., Popp, T., Gammel, B.M.: Side-Channel Leakage of Masked CMOS Gates. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 351–365. Springer, Heidelberg (2005)

25. Moradi, A., Mischke, O.: On the Simplicity of Converting Leakages from Multivariate to Univariate. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 1–20. Springer, Heidelberg (2013)
26. Nikova, S., Rijmen, V., Schläffer, M.: Secure hardware implementation of nonlinear functions in the presence of glitches. J. Cryptology **24**(2), 292–321 (2011)
27. Prouff, E., Rivain, M.: Masking against Side-Channel Attacks: A Formal Security Proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Heidelberg (2013)
28. Prouff, E., Rivain, M., Roche, T.: On the Practical Security of a Leakage Resilient Masking Scheme. In: Benaloh, J. (ed.) CT-RSA 2014. LNCS, vol. 8366, pp. 169–182. Springer, Heidelberg (2014)
29. Prouff, E., Roche, T.: Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In: Preneel, B., Takagi, T. (eds.) CHES 2011. LNCS, vol. 6917, pp. 63–78. Springer, Heidelberg (2011)
30. Quisquater, J.-J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In: Attali, S., Jensen, T. (eds.) E-smart 2001. LNCS, vol. 2140, pp. 200–210. Springer, Heidelberg (2001)
31. Rao, A.: An Exposition of Bourgain's 2-Source Extractor. Electronic Colloquium on Computational Complexity- ECCC 14(034) (2007)
32. Reparaz, O., Gierlichs, B., Verbauwhede, I.: Selecting Time Samples for Multivariate DPA Attacks. In: Prouff, E., Schaumont, P. (eds.) CHES 2012. LNCS, vol. 7428, pp. 155–174. Springer, Heidelberg (2012)
33. Rivain, M., Prouff, E.: Provably Secure Higher-Order Masking of AES. In: Mangard, S., Standaert, F.-X. (eds.) CHES 2010. LNCS, vol. 6225, pp. 413–427. Springer, Heidelberg (2010)
34. Shamir, A.: How to Share a Secret. Communications of the ACM **22**(11), 612–613 (1979)
35. Standaert, F.-X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) EUROCRYPT 2009. LNCS, vol. 5479, pp. 443–461. Springer, Heidelberg (2009)
36. Standaert, F.-X., Veyrat-Charvillon, N., Oswald, E., Gierlichs, B., Medwed, M., Kasper, M., Mangard, S.: The World Is Not Enough: Another Look on Second-Order DPA. In: Abe, M. (ed.) ASIACRYPT 2010. LNCS, vol. 6477, pp. 112–129. Springer, Heidelberg (2010)
37. von Willich, M.: A Technique with an Information-Theoretic Basis for Protecting Secret Data from Differential Power Attacks. In: Honary, B. (ed.) Cryptography and Coding 2001. LNCS, vol. 2260, pp. 44–62. Springer, Heidelberg (2001)
38. Win, E.D., Bosselaers, A., Vandenberghe, S., Gersem, P.D., Vandewalle, J.: A Fast Software Implementation for Arithmetic Operations in$GF(2^n)$. In: Kim, K., Matsumoto, T., (eds.) ASIACRYPT 1996. LNCS, vol. 1163, pp. 65–76. Springer, Heidelberg (1996)
39. Yu, Y., Standaert, F., Pereira, O., Yung, M.: Practical leakage-resilient pseudorandom generators. In: Al-Shaer, E., Keromytis, A.D., Shmatikov, V. (eds.) Computer and Communications Security, CCS 2010, pp. 141–151. ACM (2010)